A low cost RSA chip based on CRT

Min Wu, Xiaoyang Zeng, Jun Han, Yongxin Ma, Yongyi Wu, Guoquan Zhang

State Key Laboratory of ASIC & System (Fudan University), Shanghai 200433, P.R. China

Abstract:

In this paper, the authors present a VLSI design and ASIC implementation of a low cost RSA cryptosystem based on the modified Montgomery algorithm and the Chinese Remainder Theory (CRT), as well as a new scheduling scheme. The CRT technique improves data throughput, and the new scheduling scheme reduces hardware complexity. As a result, a 1024-bit modular exponentiation calculation can be performed in about 1.2 mega cycles, and the core size is less than 54K gates. With 40MHz system clock, a signature rate over 33kbps can be achieved. Using the SMIC 0.25um CMOS process, the clock frequency can be up to 125MHz and in sequence the signature rate to 100kbps.

Key words: public-key cryptosystem, RSA, CRT, Modified Montgomery algorithm, Digital Signature and Verification

1. Introduction

With the rapid development of network, people's life style is experiencing a profound change. While enjoying the great convenience, people are also in face of a gigantic challenge of the security problem of information. Among the varieties of information, personal identity is a very important one. To ensure its security, a solution, which based on the public-key infrastructure (PKI) and called identity signature & verification, has been widely used.

Among the public-key cryptosystems, RSA is the most famous one with its core operation based on modular multiplication of great integer. In 1985, P. L. Montgomery proposed an excellent algorithm called Montgomery algorithm[1] that makes the VLSI hardware implementation of modular multiplication very efficient. From then on, many kinds of architectures based on Montgomery algorithm have been proposed for different applications. For example, Gong Peijun presented a 1024-bit RSA coprocessor based on modified Montgomery algorithm[2]; Chung-Hsien Wu proposed another RSA coprocessor using CRT to improve its data throughput[3]. Both results showed their high performance, but their hardware complexities are very high, which consume 126K and 109K gates respectively. The high hardware cost holds back their usage in smart card field.

In this paper, a novel scheduling scheme is introduced, which enables a small operating unit to perform modular multiplication of two large integers; the modular exponentiation based on CRT is also adopted to speed up data throughput. As the implementation result shows, it costs less than 54K equivalent gates, but 30kbps baud-rate can be achieved at 40MHz system clock.

Before introducing the details, an assumption in the following part should be kept in mind. Let N denotes the module in RSA cryptosystem, and P, Q are the two prime factors of N, that is, N = P * Q. Then,

Assumption: Neither of P and Q will be wider than 574 bits when N is a 1024-bit integer.

The 574-bit declaration in this assumption ensures more 'P's and 'Q's, and thus more 'N's, to be chosen, and on the other hand, enables the reduction step of algorithm 1, which will be described in section 2, to be removed directly.

Note that above assumption is not made freely: As is known to all, it is recommended that the difference between P and Q in RSA cryptosystem should not be too large in order to avoid being crashed by force attack. In general cases above assumption can be met perfectly.

2. Algorithms for Implementation

2.1. Modular Multiplication

The modular multiplication used in this paper is based on the algorithm by Young Sae Kim[4]. Given two N-residues A and B, and an extra r such that $r = 2^n$, modular multiplication can be described as:

Algorithm 1 (Montgomery Multiplication)

Input: A, B, N
Output: MM (A, B, N) = $A*B*r^{-1} \bmod N$
Step1: R = 0
Step2: for i = 0 to n-1 do {
$R = R + A_iB$;
$R = R + R_0N$;
R = R/2;
}
Step 3: if(R>N) R = R – N;
Step 4: return R;

According to Young Sae Kim, the reduction step (Step 3) can be removed directly as long as the bit width of r is two bits more than that of N.

In this paper, the aforementioned assumption ensures the removal condition. As a result, both the circuit and the performance can be improved slightly.

2.2. Modular Exponentiation

There are two methods to realize the modular exponentiation. One is called L-R algorithm and the other R-L algorithm. Generally R-L algorithm can speed up the throughput about 2 times, but its corresponding cost doubles. Considering the hardware limit, L-R algorithm is adopted in this paper, as follows:

Algorithm 2 (Modular Exponentiation: L-R)

Input: M, e, N, $C = 2^{2n} \bmod N$
Output: $R = M^e \bmod N$, $0 \le R < N$
Step 1: M' = MM (M, C, N),
R = MM (C, 1, N);
Step 2: for i = n-1 to 0 do {
R = MM (R, R, N);
If (e_i == 1) then
R = MM(R, M', N);
Else
R = R;
}
Step 3: R = MM (R, 1, N);
Step 4: return R;

2.3. RSA based on CRT

The solution using CRT to speed up the RSA has been elaborated by Taek-Won Kwon[5], and it can be described briefly as following:

Algorithm 3 (Modular Exponentiation based on CRT)

Step 1: Calculate $C_P = C \bmod P$ and $C_Q = C \bmod Q$.
Step 2: Calculate the modular exponentiation $M_P = C_P^{D_P} \bmod P$ and $M_Q = C_Q^{D_Q} \bmod Q$.
Step 3: Calculate the coefficients $S_P = M_P R_P \bmod N$ and $S_Q = M_Q R_Q \bmod N$.
Step 4: Calculate $M = S_P + S_Q$, if $M \ge N$ then calculate $M = M - N$.

Note that the computation of comparison and subtraction in step 4 can be performed by a modular reduction, thus no compare or subtract circuit is required, in spite of some loss in performance.

2.4. A New Scheduling Scheme

It's obviously that the computation of steps 1, 3 and 4 in algorithm 3 cannot be performed directly using 576-bit modular multiplication unit. However, computations involved in these steps only consist of shifts and additions, which are easy to be divided-and-conquered. .Details as follow:

Look back to algorithm 1. The for-loop body can be transformed into the following form:

$$R_i = \begin{cases} R_{i-1} + BN, & if\ A[i]\&(B[0]\hat{}R_{i-1}[0]) \\ R_{i-1} + B, & if\ A[i]\&\sim(B[0]\hat{}R_{i-1}[0]) \\ R_{i-1} + N, & if\ \sim A[i]\&(B[0]\hat{}R_{i-1}[0]) \end{cases}$$

$$R_i = R_i >> 1$$

Where BN is a pre-computed integer whose value equals (B+N).

Suppose A-step denotes the addition step in above equations, and S-step denotes the shift step. Then A-step can be split into to parts:

$$R^L_i = \begin{cases} R^L_{i-1} + BN^L, & if\ A[i]\&(B[0]\hat{}R_{i-1}[0]) \\ R^L_{i-1} + B^L, & if\ A[i]\&\sim(B[0]\hat{}R_{i-1}[0]) \\ R^L_{i-1} + N^L, & if\ \sim A[i]\&(B[0]\hat{}R_{i-1}[0]) \end{cases}$$

$$R^H_i = \begin{cases} R^H_{i-1} + BN^H, & if\ A[i]\&(B[0]\hat{}R_{i-1}[0]) \\ R^H_{i-1} + B^H, & if\ A[i]\&\sim(B[0]\hat{}R_{i-1}[0]) \\ R^H_{i-1} + N^H, & if\ \sim A[i]\&(B[0]\hat{}R_{i-1}[0]) \end{cases}$$

Where the superscript of 'L' indicates the lower half part and that of 'H' stands for the higher half part, and the subscript of 'i' denotes the value in i^{th} iteration. For example, R^H_i means the half part of R in i^{th} iteration.

Note that the former step may generate a carry bit that should be propagated to the latter step. The following S-step, however, enables the carry bit to be saved for one cycle and then be taken part into the next addition of R^L, e.g. R^L_{i+1}. As a result, the for-loop in algorithm 1 can be divided into two:

for i=0 to 575 do{

$$R^L_i = \begin{cases} R^L_{i-1} + BN^L, & if\ A[i]\&(B[0]\hat{}R_{i-1}[0]) \\ R^L_{i-1} + B^L, & if\ A[i]\&\sim(B[0]\hat{}R_{i-1}[0]) \\ R^L_{i-1} + N^L, & if\ \sim A[i]\&(B[0]\hat{}R_{i-1}[0]) \end{cases}$$

$$R^L_i = R^L_i >> 1$$

}

for i=576 to (n- 577) do{

$$R^H_i = \begin{cases} R^H_{i-1} + BN^H, & if\ A[i]\&(B[0]\hat{}R_{i-1}[0]) \\ R^H_{i-1} + B^H, & if\ A[i]\&\sim(B[0]\hat{}R_{i-1}[0]) \\ R^H_{i-1} + N^H, & if\ \sim A[i]\&(B[0]\hat{}R_{i-1}[0]) \end{cases}$$

$$R^H_i = R^H_i >> 1$$

}

It is clear that the operand width can be reduce to half if above for-loop are performed in sequence. On the other hand, the selections in above split for-loops are based on the lowest bit of B and R_{i-1} nevertheless they cannot be accessed when calculating R^H_i Therefore the value of $(B[0]\hat{}R_{i-1}[0])$, which is called scanning information and signed with 'Q' in the later parts of this paper, are necessary to be saved when calculating R^L_i.

Based on above analysis, and suppose that reg_A, reg_B, reg_R and reg_Q denote the registers that contain the operators of A, B, R and Q respectively, then the new scheduling scheme can be described as:

- *MM(A_0, B_0, N_0)*: preset reg_R and reg_Q and then start MM. After calculation, store reg_R and reg_A
- *MM(A_0, B_1, N_1)*: initialize reg_R and reg_Q with the previous results in step 1 and then start MM. After multiplication, store reg_A, reg_R and reg_Q
- Perform an 1156-bit addition to get the integrity result.
- *MM(A_1, B_0, N_0)*: initialize the reg_R with the lower 576 bits of the result in step 3, as well as that of reg_A and reg_Q. And then start MM. After calculation, save the operation results.
- Perform another 1152-bit addition to get the integrity result.
- *MM(A_1, B_1, N_1)*: initialize the reg_R with the result in step5, as well as that of reg_A and reg_Q, and then start MM. After calculation, {reg_R, reg_A} is the final result of the 1024-bit modular multiplication.

3. Architecture for RSA chip

The RSA cryptosystem in this paper consists of a modular multiplier based on Montgomery algorithm and several controllers of scheduling mechanism.

3.1. Flexible Word Adder (FWA)

CSA architecture is used frequently to shorten the critical path of wide-bit-adder. A general CSA used in RSA cryptosystem is shown in fig. 1. CSA, however, cannot produce the final result. To solve this problem, many solutions have been proposed in recent publications, and additional adders are required in most cases. Because of the limit in hardware costs, a solution without additional adders is more attractive to us.

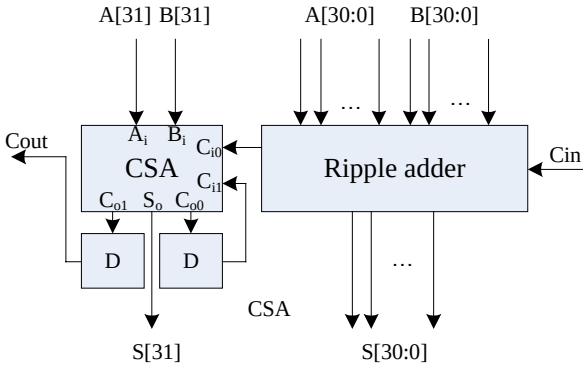

Fig. 1: Architecture of CSA in RSA

The architecture adopted in this section is called FWA, which is shown in fig. 2. The difference between fig. 1 and fig. 2 is that a func_sel signal has been used in the latter.

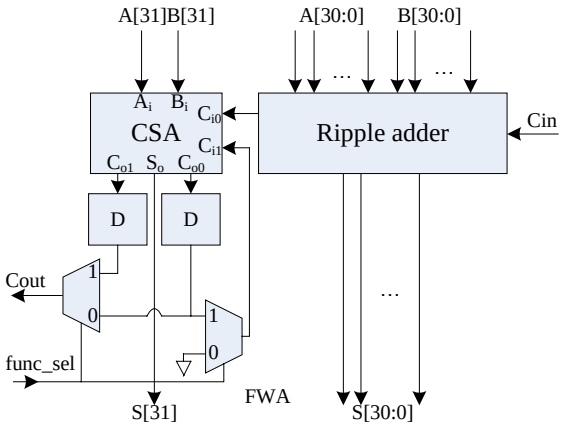

Fig. 2: Modified architecture of FWA

When func_sel is high, the FWA is unique to the CSA as in fig. 1; and when it is low, the FWA works as a ripple adder with its carry stored in a D latch (the left one in the figure). Thus the FWA can also be used to perform an integrity addition. Note that 18 clocks are needed to realize a 576-bit addition because of these latches.

Based on the FWA, the 576-bit adder can be illustrated in fig. 3.

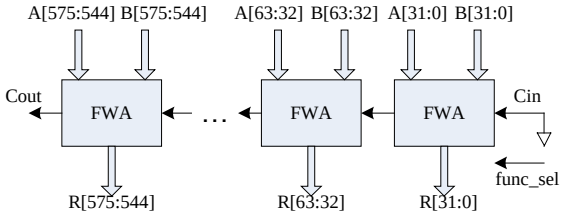

Fig. 3: 576-bit adder

3.2. Modular Multiplier (MM)

The architecture of MM can be simply illustrated in fig. 4, where q=reg_B[0] ^ reg_R[0].

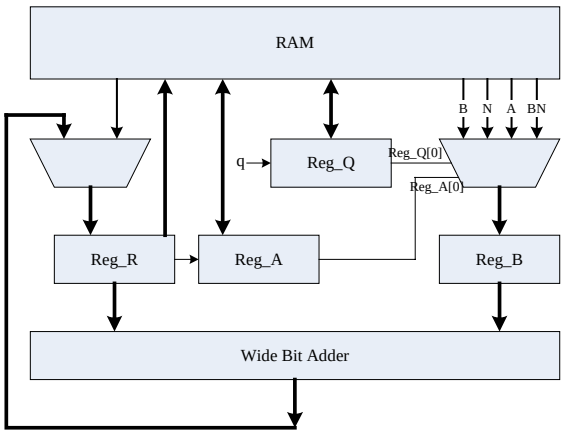

Fig.4: Architecture of MM

In fig. 4, reg_A, reg_Q and reg_R work as shifters, and they also interact with the memory depending on the algorithm's requirement. reg_B

works as a normal register to keep either operands of the addition.

Before starting modular multiplication, reg_A and reg_B should be loaded with the corresponding operands; and reg_R, as well as reg_Q, should be either reset to zero or loaded with the initial value, according to the scheduling mechanism. Once started, the higher 575 bits of the addition result (the output of the wide bit adder) are stored in reg_R, with it's 575^{th} bit being set to zero; and the lowest bit of the addition result is stored into the highest bit of reg_A after its shifting right for one bit. At the same time the exclusive-or result of reg_R[0] and reg_B[0] is also shifted into the reg_Q. After 576 iterations, reg_A contains the higher 576 bis of the result and reg_R contains the lower 576 bits, as well as reg_Q contains the scan information Q. Then the data in reg_A, reg_R and reg_Q can be stored selectively according to the scheduling mechanism in the following part.

3.3. Scheduling

There are two major scheduling schemes in this paper: the scheme of realizing 1024-bit modular multiplication and the system scheduling scheme based on CRT. They are illustrated in fig. 5 and fig. 6 respectively.

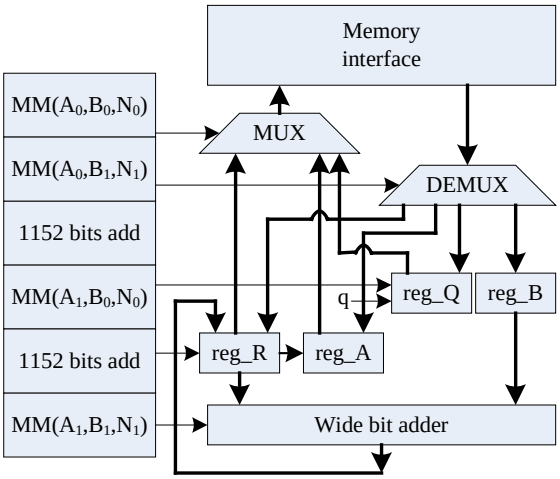

Fig. 5: Scheduling Mechanism of 1152-bit MM

In fig. 5, the right part is actually the MM architecture and the left is the scheduling controller based on the new scheduling algorithm. There are 6 steps in the algorithm, thus there are 6 controlling FSMs correspondingly. These FSMs direct the MM to accomplish storing/loading operations, modular multiplication or addition operations. After all the 6 steps are over, the final result of 1024-bit modular multiplication is produced.

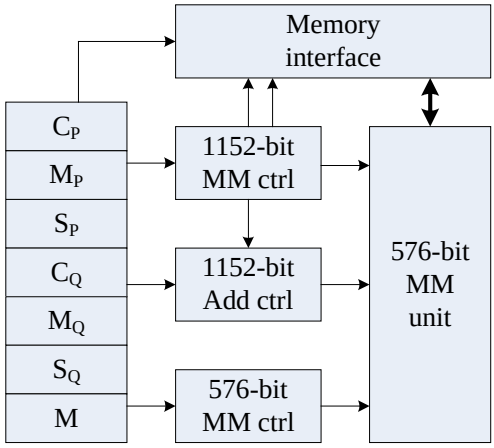

Fig. 6: 1024-bit Modular Exponentiation scheduling mechanism based on CRT

The left part in fig. 6 shows the scheduling mechanism based on CRT. These 7 steps in the algorithm are also realized by a sequence of FSMs, and these FSMs invoke sub-controllers, such as 1152-bit MM ctrl, 576-bit MM ctrl and 1152-bit add ctrl and so on, to achieve the goal of 1024-bit modular exponentiation.

4. ASIC Implementation Result and Performance Analysis

4.1. ASIC Implementation

Based on the SMIC 0.25um CMOS technology, the gate numbers of the RSA chip is less than 54K. After placing and routing, the die size is about $3x3mm^2$. Figure 7 shows the layout of the RSA chip based on CRT, and table 1 shows the comparisons among ours RSA chip and some other recent realizations with their key features. Table 1 indicates that the RSA chip in this paper has the merit of low hardware complexity but considerable throughput. So it is attractive for the applications such as smart cards.

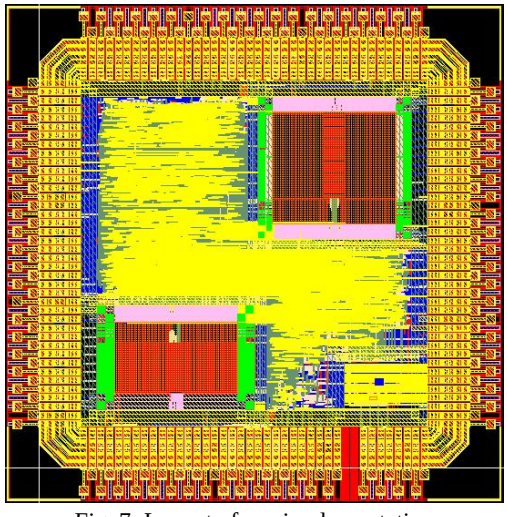

Fig. 7: Layout of our implementation

	Year	Gate	# of Clk	Tech	Clock (Hz)	Baud rate
[4]	2000	112k	2.2m	.65	50m	22.7k
[2]	2003	126k	1.1m	0.6(1)	250m	227k
[8]	1994	105k	1m	0.5(2)	40m	20k
[3]	2000	109k	.24m	0.6	150m	328k
our	2004	54k	1.2m	0.25	125m	100k

(1) 0.6um CSM

(2) 0.5um gate array

Table 1: Some RSA chips presented so far

4.2. Performance Analysis

Because of removing the reduction step and using the FWA in modular multiplication, the iteration time of once module multiplication slightly decrease to (n + 1 + n/32) cycles, where n = 576. Let T_1 = (576+1+576/32) = 595 cycles.

When performing a module exponentiation with algorithm 2, the number of modular multiplication is (2*n + 3) for the worst case and (1.5*n + 3) for the average case, therefore a 576-bit module exponentiation can be done after (2 * 576 + 3)*T_1 = 1155T_1 cycles for the worst and (1.5 * 576 + 3)*T_1 = 867T_1 clock cycles for average.

In addition, according to the new scheduling scheme, a 1024-bit modular multiplication needs (4T_1 + 2*18) plus some extra cycles for storing/loading memory. Let T_2 denotes the total cycles.

Look back to algorithm 3, and the cycles needed by every step are listed in table 2.

Generally the two steps of calculating the M_P and the M_Q will not be in the worst case at the same time, so the real consumption is a little less than the sum of all the steps in the worst cases. Experiment shows that only about 1.2 mega cycles are needed to perform a 1024-bit modular exponentiation.

Operation	Clock cycles
$C_P = C \bmod P$	$4T_1$
$M_P = C_P^{Dp} \bmod P$	$1155T_1$(worst)
	$867T_1$(average)
$S_P = M_P R_P \bmod N$	$2T_2$
$C_Q = C \bmod Q$	$4T_1$
$M_Q = C_Q^{Dq} \bmod Q$	$1155T_1$(worst)
	$867T_1$(average)
$S_Q = M_Q R_Q \bmod N$	$2T_2$
$M = (S_P+S_Q) \bmod N$	T_2+18

Table 2 Cycles in every step

5. Conclusion

In this paper, in order to design a RSA chip with a low hardware complexity but acceptable data throughput, a new 576-bit modular multiplier architecture is introduced, as well as a new scheduling method to perform 1024-bit modular multiplication with this multiplier. In addition, CRT is also introduced to improve the throughput. The implementation result shows that the hardware complexity is reduced greatly whereas the performance is comparable to most general RSA designs without CRT. All these attractive merits enable this design suitable for signature and verification applications such as smart card field.

6. Reference

[1] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, pp. 519-521, 1985

[2] Gong Peijun, "A 1024-bit RSA cryptosystem hardware design based on modified Montgomery's algorithm," *Proceedings of the 5th International Conference on ASIC 2003*, pp. 1296-1299, 2003

[3] Chung-Hsien Wu, "RSA cryptosystem design based on the Chinese Remainder theorem," *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001*, pp. 391- 395, 2001

[4] Young Sae Kim, "Implementation of 1024-bit modular processor for RSA cryptosystem," *The Second IEEE Asia and Pacific Conference on ASICs*, pp. 187-190, 2000

[5] Taek-Won Kwon, "Two implementation methods of a 1024-bit RSA cryptoprocessor based on modified Montgomery algorithm," *IEEE ISCAS*, pp. 650-653, 2001

[6] Jae-Cheol Ha, and Sang-Joe Moon, "A Design of Modular Multiplier Based on Multi-precision carry save adder", *Joint Workshop on Information Security and Cryptology*, pp. 45-52, 2000

[7] James Goodman, etc. "An Energy-Efficient Reconfigurable Public-Key Cryptography Processor," *CHES 2000*, pp. 175-190, 2000

[8] S. Ishii, K. Ohyama, and K. Yamanaka, "A single-chip RSA processor implemented in a 0.5um rule gate array," *Proc.7th Annu. IEEE Int. ASIC Conf. Exhibit*, pp.433–436, 1994

[9] Ching-Chao Yang, "A New RSA Cryptosystem Hardware Design Based on Montgomery's Algorithm", *IEEE Transactions On Circuits and Systems—II: Analog and Digital Signal Processing*, VOL.45, No. 7, July 1998